Master Informatique **Approche Objet – 4TIN706U**
Université Bordeaux **2017 – 2018**

TD N^o3 Model, View Controller

Un convertisseur de monaie

Le but de cet exercice est de réaliser une application java pour un petit convertisseur de monaie comme celle-ci :

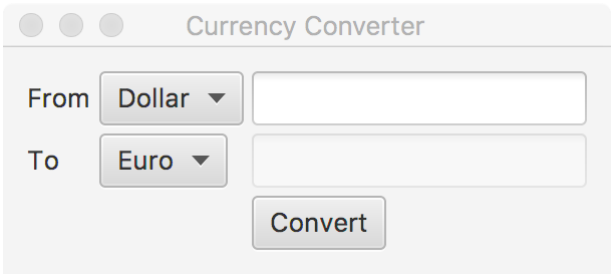

FIGURE 1 – Convertisseur

C'est un prétexte pour écrire convenablement un projet en respectant le *design patern Model-View-Controller*.

Nous invitons l'étudiant à suivre pas-à-pas cette feuille de TD pour arriver au résultat.

1 Configuration d'Eclipse

— Paramétrer Eclipse pour qu'il utilise le JDK 8
  — Ouvrir les préférences et sélectionner la partie Java — Installed JREs.
  — Cliquer sur le bouton Add... pour ajoutez le JDK 8 puis sur Standard VM et sélectionner le dossier contenant le JDK 8.
  — Supprimer les autres JREs et JDKs.

2 Créer une interface homme-machine vide

1. Créer un *package* `org.converter.view` et y ajouter une classe `View` pour réaliser la présentation graphique. La laisser vide pour l'instant.
2. Créer un *package* `org.converter.model` et une classe `Model` pour réaliser le module fonctionnel du convertisseur. La laisser vide pour l'instant.
3. Créer un *package* `org.converter.controller` et une classe `Controller` pour réaliser l'interface. La laisser vide pour l'instant.
4. Créer la classe `Main` dont le code est le suivant :

Listing 1 – Main.java

```
import org.converter.controller.Controller;
```

```java
public class Main {

        public static void main(String[] args) {
                Controller controller = new Controller();
        }

}
```

3 Appliquer le *design pattern* `Singleton` aux trois classes `Model`, `View` et `Controller`

On appliquera le *design pattern* `Singleton` pour s'assurer que chacune de leurs instances sont uniques.

Le code de la classe `Main` sera dorénavant le suivant :

Listing 2 – Main.java

```java
import org.converter.controller.Controller;

public class Main {

        public static void main(String[] args) {
                Controller.getInstance();
        }

}
```

4 Créer une interface graphique

Le but est de produire maintenant la fenêtre présentée en 1.

La class Main étend la classe `Application` et sera implémentée ainsi :.

Listing 3 – Main.java

```java
import org.converter.controller.Controller;

import javafx.application.Application;
import javafx.stage.Stage;

public class Main extends Application {

        static Controller controller;

        public static void main(String[] args) {
                controller = Controller.getInstance();
                launch(args);
        }

        @Override
        public void start(Stage primaryStage) throws Exception {
                controller.start(primaryStage);
        }
```

```
}
```

Dans la classe `View`, créer l'ensemble de l'application graphique
- Fenêtre principale : `stage`
- Contenant principal : `scene`
- Contenants : `BorderPane`, `HBox`, `VBox`, `StackPane`, `GridPane`, `FlowPane`, etc.
- Différents objets graphiques utiles
  - `Label` : Zone de texte
  - `TextField` : Zones d'édition de texte
  - `Button` : Boutons
  - `MenuButton`, `MenuItem` : Menus

  Ajouter les éléments graphiques dans le contenant .

5 Créer une interface homme-machine

5.1 Faire communiquer la vue et le modèle avec le contrôleur

- Vue :
  Ajouter une méthode `setOnAction(this)` à la classe `View`. Cette méthode se contentera d'appeler les méthodes `setOnAction(EventHandler<ActionEvent> eventHandler)` des différents objets graphiques interactifs (boutons, `textfields`).
  Appeler cette méthode lors de la construction de l'instance du contrôleur.
- Contrôleur :
  La classe `Controller` implémente maintenant l'interface `implements EventHandler<ActionEvent>` qui contient la seule méthode `public void handle(ActionEvent event)`. La méthode est évoquée quand un évènement survient (typiquement quand l'utilisateur appuie sur un bouton).
  Remarquer que le modèle et la vue sont en capacité de recevoir des données par l'intermédiaire des méthodes du contrôleur, mais qu'il n'y a aucune communication directe entre les deux.

5.2 Rendre la vue interactive

Implémenter la méthode `void drawData(String data)` en faisant afficher les données dans le `textField`.

L'effet des boutons est déjà réalisé par l'intermédiaire de la méthode `setOnAction`.

5.3 Rendre le modèle interactif

Créer une propriété `String data` dans le modèle qui sera l'élément retourné par le *getter* `String getData()`.

Le but de la suite sera d'affecter une valeur correcte à `data` en fonction des données passées à la méthode `void executeCommand(String command)`.

6 Écrire le modèle

Bon, c'est bien beau tout ça, mais on n'a toujours aucune fonctionnalité. Nous allons les ajouter.